enova365 dla biznesu systemowy rozwój firmy

enova365

ERP software for management.
Strengthening and growing with your business.
www.enova.pl

Table of content

enova365.Integrator

What is the enova365.Integrator module and what is it used for?

The **enova365.Integrator** module provides the application programming interface (API) for enova365. It enables external IT systems to retrieve data from enova365, as well as to add and update them. With the module, enova365 is able to support external systems of any hardware platform. Communication between the systems is established by the Web Service through the HTTP protocol and with the SOAP-XML format. The structure of data retrieved or transferred in the SOPA 1.1 message is entirely definable (within the XML format). The data format is defined through proven mechanisms applied in enova365.EDI.

Integration of two systems usually takes the form of development of a service (application) to communicate with each of the systems. enova365.Integrator ensures communication of the integrating service with the enova365 installation.

The module offers only two general methods: **Get** (for retrieving data) and **Update** (for adding/updating data). In the parameters of these methods, we specify the type of data to be retrieved/updated and the definition of the XML data format to be applied. This way, integration with the external system is not restricted by rigidly defined data formats.
enova365.Integrator employs the same mechanisms for generating and processing XML files which are used in enova365.EDI, whereas here, the type of objects to be added/modified in not restricted. Therefore, simplifying to a certain extent, it can be referred to as extended enova365.EDI with access through the Web Service.

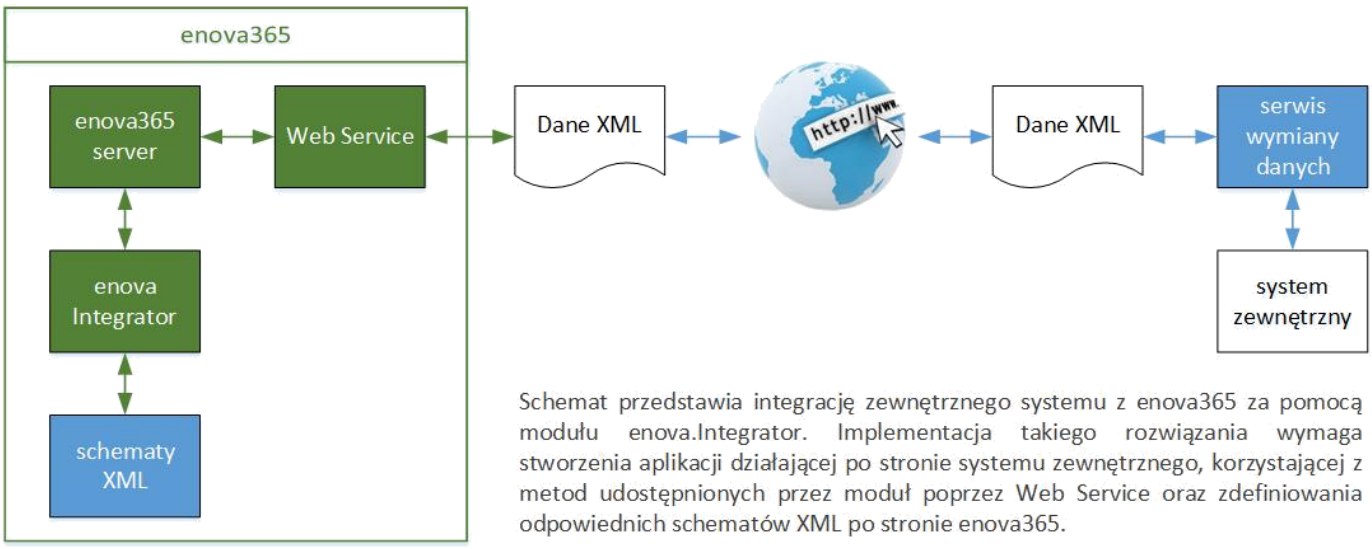
Installation

The module can be used in the enova365 multi installation, built to work through the HTML interface, i.e., based on the enova365 server and an IIS website sharing the application. The Web Service employed by enova365.Integrator uses the same website. The module is installed only by preloading to the catalogue from which it will be loaded by the enova365 server (or to the database). The WCF services must be activated in the functions of the Windows system.
The easiest way to verify whether the installation is ready to be used is to log in to enova365 in the browser and select the \Systemowe\Integrator tab in the configuration window. If the tab is visible, the module has been loaded.

How is the module built

The module is based on the following mechanisms:

- Web Service – operating on the enova365 server.
- Definitions of own XML messages (schemas) for exchanging data. Definition of messages according to the same rules is also used in the Electronic Document Interchange, EDI.
- Two defined methods: Get, Update (requested with the corresponding parameters), used respectively for retrieving and updating data.
- Additional tables: SystemyZewn (definitions of external systems), ZadaniaZewnSynch (synchronisation tasks), RelacjeZewn (relations between local and external objects). The tables can be used but are not necessary for operation of the Get and Update methods.



Schemat przedstawia integrację zewnętrznego systemu z enova365 za pomocą modułu enova.Integrator. Implementacja takiego rozwiązania wymaga stworzenia aplikacji działającej po stronie systemu zewnętrznego, korzystającej z metod udostępnionych przez moduł poprzez Web Service oraz zdefiniowania odpowiednich schematów XML po stronie enova365.

Web Service

The Web Service is launched on the enova365 server. Its address has the following form: **http://SERVER_NAME:SERVER_PORT/Business/MethodInvokerService.svc** and is necessary for exposure of the interface for communication of the client application with the server. The availability and communication with the service can be checked through entering of the address in the browser.

In the .NET environment, we can generate the corresponding classes (called proxy classes) from the aforementioned address, to ensure calling up of the methods provided by the Web Service.

The aforementioned Web Service provides one method with the following signature:

InvokeServiceMethodResult InvokeServiceMethod(ServiceMethodInvokerParams invokerParams)

For this methods, a parameter of the ServiceMethodInvokerParams is sent, specifying the database for which the request is valid, and the target operation to be performed (Get or Update). We also specify parameters to be sent to the requested target method, i.e., Get or Update. They are specified in the form of a dictionary with key (parameter name) – value pairs.

```
public class InternalServiceMethodInvokerParams
{
  public string DatabaseHandle { get; set; } // database name
  public string Operator { get; set; } // operator (login)
  public string Password { get; set; } // operator's password
  public string ServiceName { get; set; } // name of the service requested: "enova.Integrator.IIntegrator, enova.Integrator"
  public string MethodName { get; set; } // name of the method requested: "Get" or "Update"
  public Dictionary<string, object> MethodArgs { get; set; } // parameters sent to the
                                    method requested // in the form of pairs (key – value)
}
```

The result of operation of the above method is an object of the InvokeServiceMethodResult type, where the object of the requested method of the service is returned in the ResultInstance property

(Get, Update). The object will be an XML file. The file will contain the retrieved data in the defined format or IDs of the added/updated data.

The structure of the file is presented below.

Definitions of XML messages

Messages to be used by the Integrator are defined in enova365 in Konfiguracja, the Systemowe->Integrator tab. We can use 7 predefined messages (Towary, Kontrahenci, DokumentyHandlowe, DokumentyEwidencji, ZadaniaCRM, RaportyESP, DokumentyDMS) or create own ones. The standard definitions support retrieving and updating of data as well as creating of single entries. The single message definition specifies the XML format for the data to be returned with the method Get used or the format for the data to be delivered for updating (with the Update method). For the enova365.Integrator, the relevant parameters are those specifying the DataType and the name of the defined message. Those two fields are required for calling the methods of the module.

In the standard messages, the following logic has been applied: each message contains the Identyfikator (ID) field for the object's Guid to be placed during data export. If the field is present in the imported file and is filled, the importing mechanism performs the search for a corresponding object and updates it with the data supplied. If the Identyfikator field is not present in the file, a new object is added.

For detailed information on defining messages, refer to the documentation of enova.EDI.

Methods of the Integrator

The enova365.Integrator module (service) provides two methods:

```
public interface IIntegrator
{
  object Get(GetParams pars);
  object Update(UpdateParams pars);
}
```

The first one is used for retrieving data from enova365. The second method is used for updating/adding new data in accordance with the defined XML schema.

Retrieving data

Data are retrieved with the Get method. The method receives a parameter class specifying the parameters for retrieving. It is a list of pairs: key (parameter name) – value.

```
public class GetParams
{
  public string TableName { get; set; } // name of the table from which data are to be retrieved, e.g. "DokHandlowe", "Towary"
  public string SchemaName { get; set; } // name of the XML schema (definition), e.g. "Towary"
```

public string Condition { get; set; } // optional condition filtering the data retrieved, built in accordance with the rules used for building filters for lists, e.g. "Kod = 'AG001'" public int RowCount { get; set; } // specifies the number of records to be retrieved, by default: 100

```
public DateTime LastDate { get; set; } // date of modification of the last record.
                    // If specified, the first package of data modified after the given date (returned in the class of results)
public int LastId { get; set; } // ID of the record returned last so far … necessary for retrieving the next data package (will be returned in the class of results)

// From version 12.3
public int Search { get; set; } // Method of searching for data to be returned from the Get method. Options: 1, 2, 3 (described below)
public string ExternalSystem { get; set; } // Name of the external system, for which data for synchronisation
are created public string ActionName { get; set; } // Name of action for the task to be synchronised
public string SchemaParam { get; set; } // Parameter sent to the XML schema, with access through
                    GetFromDict("EXTERNAL_PARAM_Integrator"), // can control the schema's (definition's) method of operation
// From version 15.0
public string ResultDataFormat { get; set; } // Format of the resulting data: 1 (by default) – complete XML file as CDATA in GetResult, 2 – tag combined with GetResult.
}
```

The Get method returns the result in the form of an XML file in accordance with the schema http://www.enova.pl/Schemas/GetResult.xsd. It contains data concerning the objects retrieved in the XML form and contained in fields of the CDATA type. An example of the file is shown below.

Methods of searching for the data returned

The method of searching for data and the sequence of retrieving them is specified by the Search parameter in the GetParams parameter class. It has the following values:

- *Search = 1* – (default value) data are retrieved in the order of the default key for the table specified by the TableName parameter.
- *Search = 2* – data are retrieved according to the date of the last change recorded in the object's change history (ChangeInfos); in this case, the additional parameter LastDate is applicable, specifying the date of the last change from which are data are to be retrieved.
- *Search = 3* – only those data will be retrieved which have entries in the table with synchronisation tasks (ZadaniaZewnSynch); in this case, the additional parameters ExternalSystem, ActionName are applicable.

Search = 1 (base key)

enova365.Integrator performs the search and returns data from the table specified in TableName, in order according to the base key, meaning the one defined as TableName_Podstawowy. For example: for the parameters *TableName = "Towary", RowCount = 5, Search = 1*, the system will return 5 top records directly from the Towary (Goods) table, in the order of the code, because this is how the base key was defined for this table (Towary_Podstawowy).

ID	Kod	Nazwa	LastChangeInfo
36	AG001	sprzęgło kpl. Ford Capri AGM	16.04.2018 15:55:21
17	AG002	tarcza sprzęgła Ford Capri 180mm AGM	07.03.2018 08:59:04
6	AG003	docisk sprzęgła Ford Capri 180mm AGM	07.03.2018 09:03:08
13	AG004	łożysko wyciskowe Ford Capri AGM	07.03.2018 09:09:56
22	BONUS	rabat okresowy	07.03.2018 09:06:37
12	BS001	klocki hamulcowe prz. Cadillac CTS 2002-2007 BrakeStar	16.04.2018 16:02:14
10	BS002	klocki hamulcowe prz. Buick LaCrosse 2005-2009 BrakeStar	22.06.2017 15:12:02
19	BS003	tarcze hamulcowe prz. 300mm Cadillac CTS 2002-2007 BrakeStar	07.03.2018 09:06:37
37	BX001	sprzęgło kpl. Ford Capri BXM	07.03.2018 09:08:21
18	BX002	tarcza sprzęgła Ford Capri 180mm BXM	07.03.2018 09:09:58

Search = 2 (change history in ChangeInfos)

enova365.Integrator performs the search and returns data of the type specified in TableName, but with indication in the table of object change history. For example: for the parameters *TableName = "Towary", RowCount = 5, Search = 2*, LastDate = 2018-03-20, the system will return 5 top records of the Towary type which have been modified after 2018-03-20 (with entry in the ChangeInfos table).

13	AG004	łożysko wyciskowe Ford Capri AGM	07.03.2018 09:09:56
14	BX004	łożysko wyciskowe Ford Capri BXM	07.03.2018 09:09:58
18	BX002	tarcza sprzęgła Ford Capri 180mm BXM	07.03.2018 09:09:58
32	DE001	akumulator żelowy 120 Ah 700A DexoElectro	22.03.2018 10:28:01
34	DE002	akumulator żelowy 80 Ah 500A DexoElectro	22.03.2018 10:28:30
15	PK10	przesyłka kurierska do 10 kg	06.04.2018 14:26:53
16	PK20	przesyłka kurierska do 20 kg	06.04.2018 14:27:09
36	AG001	sprzęgło kpl. Ford Capri AGM	16.04.2018 15:55:21
12	BS001	klocki hamulcowe prz. Cadillac CTS 2002-2007 BrakeStar	16.04.2018 16:02:14
1	VM003	pompa paliwa Aston Martin Lagonda	16.04.2018 16:25:36

Search = 3 (synchronisation tasks)

enova365.Integrator performs the search and returns data of the type specified in TableName, but with indication in the table of tasks to be synchronised. For example: for the parameters *TableName = "Towary", RowCount = 5, Search = 3, ExternalSystem="MA_22", ActionName="Add",* the system will return 5 top records of the Towary type which are flagged for synchronisation for the MA_22 system (with entry in the ZadaniaZewnSynch table).

ID	Akcja	Zapis tabela	System zewn	Data	Zapis
1	Add	Towary	eSklep - MA_22	16.04.2018 16:38:35	001da03b-6aeb-4c83-a7f6-68b66dde0dd7
3	Add	Towary	eSklep - MA_22	16.04.2018 16:38:41	3999a53e-e399-4f95-8a3e-f8769f1e5ec7
5	Add	Towary	eSklep - MA_22	16.04.2018 16:38:46	902d7b14-0ccc-43e7-ab2c-e6e83cdc88a0
7	Add	Towary	eSklep - MA_22	16.04.2018 16:38:53	d85d9473-a731-4ddb-843d-e4e210c108c5
9	Add	Towary	eSklep - MA_22	16.04.2018 16:38:58	97fd9da4-f15d-40e7-9ed9-03bf720b3321
11	Add	Towary	eSklep - MA_22	16.04.2018 16:39:04	9fe6f27d-ac82-4920-85f4-cb75c4fcb1be
13	Delete	Towary	eSklep - MA_22	16.04.2018 16:40:13	0caf46b9-2af3-4a62-ab03-1a7a9d2d201c
15	Add	Towary	eSklep - MA_22	16.04.2018 16:40:29	14fb9799-266c-4496-b2ac-5e7d6fc6535b
17	Update	Towary	eSklep - PR_01	16.04.2018 16:41:26	9eb0fbd7-3b7a-4799-9244-a4e7dd4b04dd
19	Add	Towary	eSklep - MA_22	16.04.2018 16:41:33	bede8458-a7e7-4aea-b7eb-221c5ac40f1a

Removing synchronisation tasks

If we retrieve data in accordance with the synchronisation tasks, the respective synchronisation tasks are to be removed after the retrieval. The tasks should be removed by the service retrieving the data. For this, the Update method can be executed for the ZadaniaZewnSynch table with indication of the XML PoSynchronizacji schema used for this purpose. This definition removes tasks identified according to the guids of objects they applied to and creates relations between synchronised objects, i.e., creates entries in the RelacjeZewn table.

Examples

An example of using the Get method for retrieving data is shown below.

```
using (MethodInvokerServiceClient myClient_ = new MethodInvokerServiceClient())
{
  var Params = new ServiceMethodInvokerParams
```

```
  {
    DatabaseHandle = "enova_demo",
    Operator = "Integrator",
    Password = "enova",
    ServiceName = "enova.Integrator.IIntegrator, enova.Integrator",
    MethodName = "Get",
    MethodArgs = new Dictionary<string, object>
    {
     { "TableName", "Towary" },
     { "SchemaName", "Towary" },
     { "Search", 1 },
     { "Condition", "Kod='AG001'" /*no need to specify*/},
     { "RowCount", 20 /*no need to specify, by default = 100*/ },
     { "LastDate", "" /*needed when retrieving modified data in packages*/ }, {
     "LastId", 0 /*needed when retrieving data in packages*/ }
    }
  };

  InvokeServiceMethodResult result = myClient_.InvokeServiceMethod(Params); // result (object), including data with exceptions,
  etc. ...
  var xmlResult = result.ResultInstance; // this is already the specific result (XML) returned by the requested Get method;
}
```

The method will return the retrieved data in the form of an XML file (included in result.ResultInstance):

```
<?xml version="1.0" encoding="utf-16"?>
<GetResult>
  <LastRow>
    <Id>22</Id>
    <Date>0001-01-01T00:00:00</Date>
    <End>false</End>
  </LastRow>
  <Rows>
    <Row>
    <Xml><![CDATA[<?xml version="1.0" encoding="utf-8"?>
     <Towar>
      <Identyfikator>9eb0fbd7-3b7a-4799-9244-a4e7dd4b04dd</Identyfikator>
      <Typ>Product</Typ>
      <Kod>AG001</Kod>
      <Nazwa>clutch cpl. Ford Capri AGM</Nazwa>
      <Jednostka>pc</Jednostka>
      (... other fields ...)
     </Towar>]]>
    </Xml>
    </Row>
    <Row>
    <Xml><![CDATA[<?xml version="1.0" encoding="utf-8"?>
     <Towar>
       (... fields ...)
     </Towar>]]>
    </Xml>
    </Row>
   </Rows>
</GetResult>
```

The LastRow section contains details of the last returned record, the Rows section contains details of the objects retrieved.

An example of retrieving all goods in packages of 20 is shown below. In the loop, we check the ID of the last record returned (LastRow/ID) and the parameter showing whether all records have already been retrieved (LastRow/End).

```
var client = new Service.MethodInvokerServiceClient();
var methodParams = new Service.ServiceMethodInvokerParams()
{
  DatabaseHandle = "enova_demo",
  Operator = "Integrator",
  Password = "enova",
  ServiceName = "enova.Integrator.IIntegrator, enova.Integrator",
  MethodName = "Get",
  MethodArgs = new Dictionary<string, object> {
   {"TableName", "Towary" },
   { "SchemaName", "Towary" },
   { "Search", 1 },
   { "Condition", "" }, {
   "RowCount", 20 }, {
   "LastId", 0 }
  }
};

var end = false;
to
{
  var methodResult = client.InvokeServiceMethod(methodParams);
  var result = methodResult.ResultInstance as string;   // here, we have a result we can process
  XmlDocument xd = new XmlDocument();
  xd.LoadXml(result);
  methodParams.MethodArgs["LastId"] = int.Parse(xd.SelectSingleNode("GetResult//LastRow//Id").InnerText);
  end = bool.Parse(xd.SelectSingleNode("GetResult//LastRow//End").InnerText);
}
while (!end);
```

Similarly, all goods modified in the last month can be retrieved. Instead of LastId, we will use the LastDate parameter.

```
  // first value
```

```
  MethodArgs = new Dictionary<string, object> {
    ...
    { "LastDate", DateTime.Today.AddMonths(-1) }
  }

  // change in loop
  methodParams.MethodArgs["LastDate"] = int.Parse(xd.SelectSingleNode("GetResult//LastRow//Date").InnerText);
```

Updating/adding data

Data can be added/updated with the Update method. The method receives a parameter class containing an XML file with the data to be processed.

```
public class UpdateParams
{
  public string TableName { get; set; } // table name
  public string SchemaName { get; set; } // name of the XML schema
  public string Data { get; set; } // data in the XML form to be processed by the defined message (schema)

  // From version 12.3
  public string ExternalSystem { get; set; } // Parameter sent to the XML schema, with access through
                    GetFromDict("EXTERNAL_PARAM_ExternalSystem"), // provides information on the external system for which the request
                    applies
  public string SchemaParam { get; set; } // Parameter sent to the XML schema, with access through
                    GetFromDict("EXTERNAL_PARAM_Integrator"), // can control the schema's (definition's) method of
                    operation

  // From version 15.0
  public string ResultSchemaName { get; set; } // Parameter sent to the XML schema. If specified, the result returned by the Update method will contain the data of the created
                    // or modified object, generated with the indicated schema. They will be located in the <Xml> tag of the result file.
                    // With the parameter, the data of the object created can be retrieved together with information on execution of the Update
                      method
}
```

The Data parameter (data to be updated) is sent in the form of an XML file in accordance with the schema http://www.enova.pl/Schemas/UpdateParamsData.xsd.

An example of adding 2 goods with specification of code and name is shown below.

```
using (MethodInvokerServiceClient myClient_ = new MethodInvokerServiceClient())
{
  var Params = new ServiceMethodInvokerParams
  {
    DatabaseHandle = "enova_demo",
    Operator = "Integrator",
    Password = "enova",
    ServiceName = "enova.Integrator.IIntegrator, enova.Integrator",
    MethodName = "Update",
    MethodArgs = new Dictionary<string, object>
    {
      { "TableName", "Towary" },
      { "SchemaName", "Towary" },
      { "Data", @"<?xml version=""1.0""
      encoding=""utf-16""?> <UpdateParamsData>
        <Rows>
          <Row>
            <Xml><![CDATA[<?xml version=""1.0""
              encoding=""utf-8""?> <Towar>
                <Kod>AG100</Kod>
                <Nazwa>clutch cpl. Jaguar Sovereign
                AGM</Nazwa> <Jednostka>pc</Jednostka>
              </Towar>]]></Xml>
          </Row>
          <Row>
            <Xml>(data of the second
          product...)</Xml> </Row>
        </Rows>
      </UpdateParamsData>" }
    }
  };
  InvokeServiceMethodResult result = myClient_.InvokeServiceMethod(Params); // result (object), including data with exceptions, etc. ...
  var xmlResult = result.ResultInstance; // this is already the specific result (XML) returned by the requested Update method;
}
```

The Update method returns the result in the form of an XML file in accordance with the schema http://www.enova.pl/Schemas/UpdateResult.xsd. The example below shows the result after processing of two objects, with empty Xml tag, meaning the case without specification of ResultSchemaName in the parameters of the method.

```
<?xml version="1.0" encoding="utf-16"?>
<UpdateResult>
 <Rows>
  <Row>
   <No>1</No>
   <ID>24</ID>
   <Guid>f26aa399-b520-4865-8fbc-f132b20921e1</Guid>
   <Xml><![CDATA[]]></Xml>
   <Message ></Message>
   <MessageType>Info</MessageType>
  </Row>
  <Row>
   <No>2</No>
   <ID>25</ID>
   <Guid>40c97751-2a36-486d-a795-0f8b76534eaf</Guid>
   <Xml><![CDATA[]]></Xml>
   <Message ></Message>
```

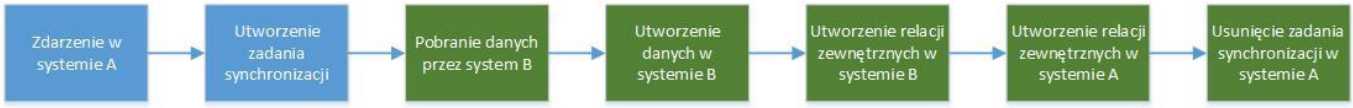
<MessageType>Info</MessageType>
</Row>

</Rows>
</UpdateResult>

The Row section contains the subsequent number of the added/updated object (in accordance with the sequence in the XML file sent as a parameter of the Update method), its Guid, ID and error message, if present.

Complete process of retrieving and saving data

The process of retrieving data is recommended to be organised with the use of definitions of external systems, synchronisation tasks and external relations. Then, the process flow will be as follows:



- Event in the source system (e.g., issuing of a document) results in a synchronisation task. Creating of the task should be programmed by yourself, e.g., through tasks.
- The external system retrieves data covered by the synchronisation tasks through the remotely executed Get method.
- The external system records the retrieved data in its database. If the enova365 installation is involved, the locally executed Update method is performed, to which the data retrieved in the Data parameter are sent.
- The external system creates records in its database, binding the objects in both systems. If enova365 is involved, records are made in the RelacjeZewn table.
- The external system removes the synchronisation tasks in the source installation, executing remotely the Update method. Removal of data creates records in the RelacjeZewn table in the source system at the same time.

The standard task **Systemy zewnętrzne/Pobieranie danych (External systems/Retrieving data)**, available from version 15.3 in the main menu of the system with an active enova365.Integrator module, used for retrieving data from a different enova365 installation, is operated in accordance with the scheme described above.

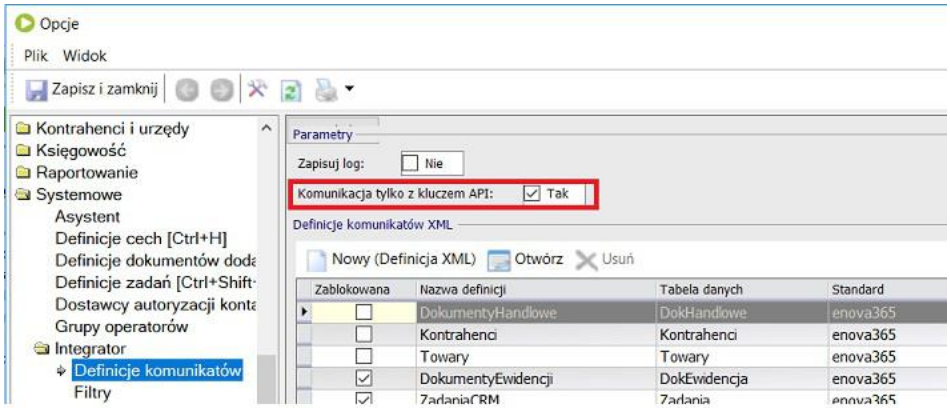
Restricting access to data

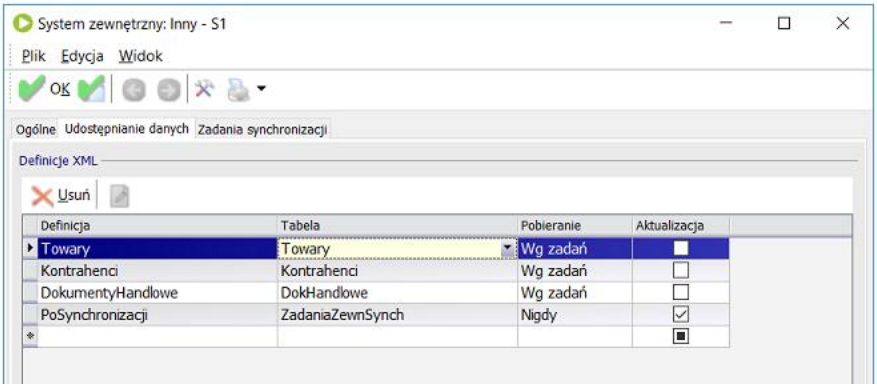
Activation of enova365.Serwer with the Integrator module and defined Web site means that data are made available for external applications which can retrieve them with the Get method. Naturally, retrieval of data from outside is possible when we provide the external application with an operator (delivering the login and password) to be used for logging to the database. The second condition necessary for retrieving data is the existence of an unblocked XML definition which will generate such data. As a standard, access to data is protected by the rights of the operator used for logging by the external application.

From version 15.3 on, enova365 provides additional tools for transparent setting up of access to data through enova365.Integrator.

For this, the external system's definition is used. The configuration is performed as follows:

- We activate the parameter **Komunikacja tylko z kluczem API** (Communication with API key only). Now, for any request for retrieving/updating data in the *ExternalSystem* parameter, the API key must be supplied, delivered by the party sharing the data. Before executing the request, the system will verify the correctness of the key as described below. With this setting, only those data can be retrieved which are openly specified in the configuration of the system sharing the data.
- We define the external system corresponding to the installation retrieving data through enova365.Integrator. We activate the parameter **Udostępnianie danych** (Sharing data), which activates a tab with the same name. Upon activation of the parameter, the field **Identyfikator własny** (Own ID) is shown (read-only), displaying the guid of the defined system. This is the API key to be shared with the party retrieving the data. Thus, the key is unique for every external installation with a separate external system definition.
- We configure the access to data in the tab **Udostępnianie danych** (Sharing data). The row in this tab provides data from the specified table, in the form generated by the indicated schema. It also specifies the allowed mode for retrieving data, corresponding to the value of the parameter *Search*, i.e., **Zawsze** (Always) (*Search* = 1,2 3), or only **Wg zadań** (Task based) (*Search* = 3). In other words, the row contains a set of allowed parameters of the *Get* method.





With this configuration, the logic for processing of the request for retrieving data will be as follows:

- The system verifies whether an external system with a guid complying with the API key sent in the parameter *ExternalSystem* is defined.
- If not, the request is rejected as not authorised.

If it is defined, the system analyses the parameters of the request: table name, schema name and retrieving mode. The system checks in the external system definition
- searched with the API key whether

such a set of parameters is allowed, i.e., whether it is represented by a row in the tab **Udostępnianie danych** (Sharing data).
- If the set of parameters is allowed, the request will be performed.

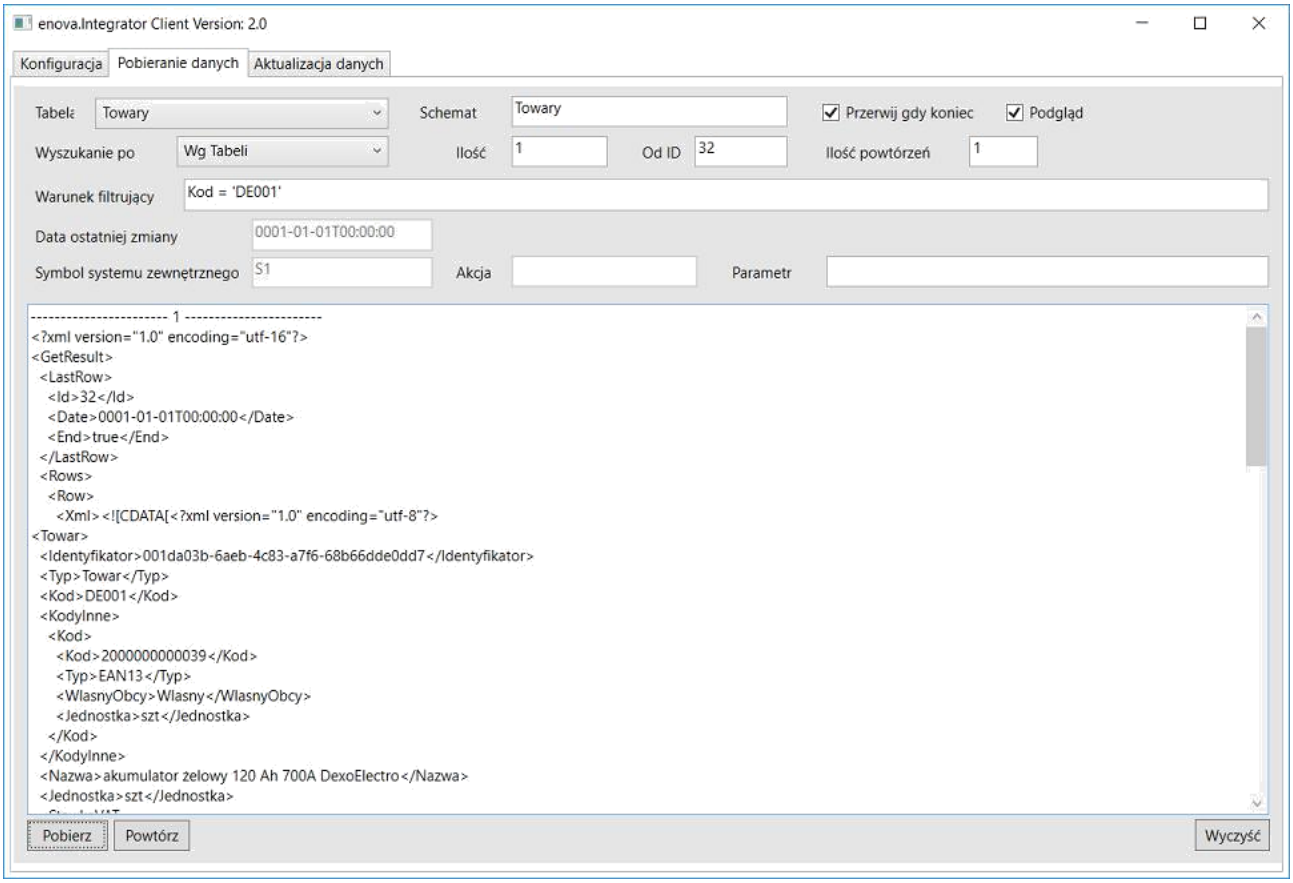
SOAP POST example

An example of a complete structure sent with SOAP 1.1 is shown below.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <InvokeServiceMethod xmlns="http://tempuri.org/">
   <invokerParams xmlns:a="http://schemas.datacontract.org/2004/07/Soneta.Net.Types" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <ClientAddress i:nil="true" xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types" ></ClientAddress>
    <ContextHandle i:nil="true" xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types" ></ContextHandle>
    <DatabaseHandle xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types">Moto_150</DatabaseHandle>
    <IpFilters i:nil="true" xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types" ></IpFilters>
    <MethodArgs xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types"
     xmlns:b="http://schemas.microsoft.com/2003/10/Serialization/Arrays"> <b:KeyValueOfstringanyType>
      <b:Key>Search</b:Key>
      <b:Value i:type="c:int" xmlns:c="http://www.w3.org/2001/XMLSchema">1</b:Value>
     </b:KeyValueOfstringanyType>
<b:KeyValueOfstringanyType>
<b:Key>TableName</b:Key>
      <b:Value i:type="c:string" xmlns:c="http://www.w3.org/2001/XMLSchema">Towary</b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>SchemaName</b:Key>
      <b:Value i:type="c:string" xmlns:c="http://www.w3.org/2001/XMLSchema">Towary</b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>Condition</b:Key>
      <b:Value i:type="c:string" xmlns:c="http://www.w3.org/2001/XMLSchema" ></b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>RowCount</b:Key>
      <b:Value i:type="c:int" xmlns:c="http://www.w3.org/2001/XMLSchema">10</b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
 <b:Key>ActionName</b:Key>
      <b:Value i:type="c:string" xmlns:c="http://www.w3.org/2001/XMLSchema" ></b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>ExternalSystem</b:Key>
      <b:Value i:type="c:string" xmlns:c="http://www.w3.org/2001/XMLSchema" ></b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>LocalRequest</b:Key>
      <b:Value i:type="c:boolean" xmlns:c="http://www.w3.org/2001/XMLSchema">false</b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>LastDate</b:Key>
      <b:Value i:type="c:dateTime" xmlns:c="http://www.w3.org/2001/XMLSchema">0001-01-01T00:00:00</b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>LastId</b:Key>
      <b:Value i:type="c:int" xmlns:c="http://www.w3.org/2001/XMLSchema">18</b:Value>
     </b:KeyValueOfstringanyType>
     <b:KeyValueOfstringanyType>
      <b:Key>SchemaParam</b:Key>
      <b:Value i:type="c:string" xmlns:c="http://www.w3.org/2001/XMLSchema"
     ></b:Value> </b:KeyValueOfstringanyType>
    </MethodArgs>
    <MethodName xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types">Get</MethodName>
    <Operator xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types">Integrator</Operator>
    <Password xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types">enova</Password>
    <ServiceName xmlns="http://schemas.datacontract.org/2004/07/Soneta.Types">enova.Integrator.IIntegrator,
    enova.Integrator</ServiceName> <a:ConnectionInfo i:nil="true" ></a:ConnectionInfo>
   </invokerParams>
  </InvokeServiceMethod>
 </s:Body>
</s:Envelope>
```

Test software (client)

The operation of the methods described can be tested with the enova.Integrator.Client application included in the package. The application enables retrieving and updating of data, presents the XML created and allows for editing them.



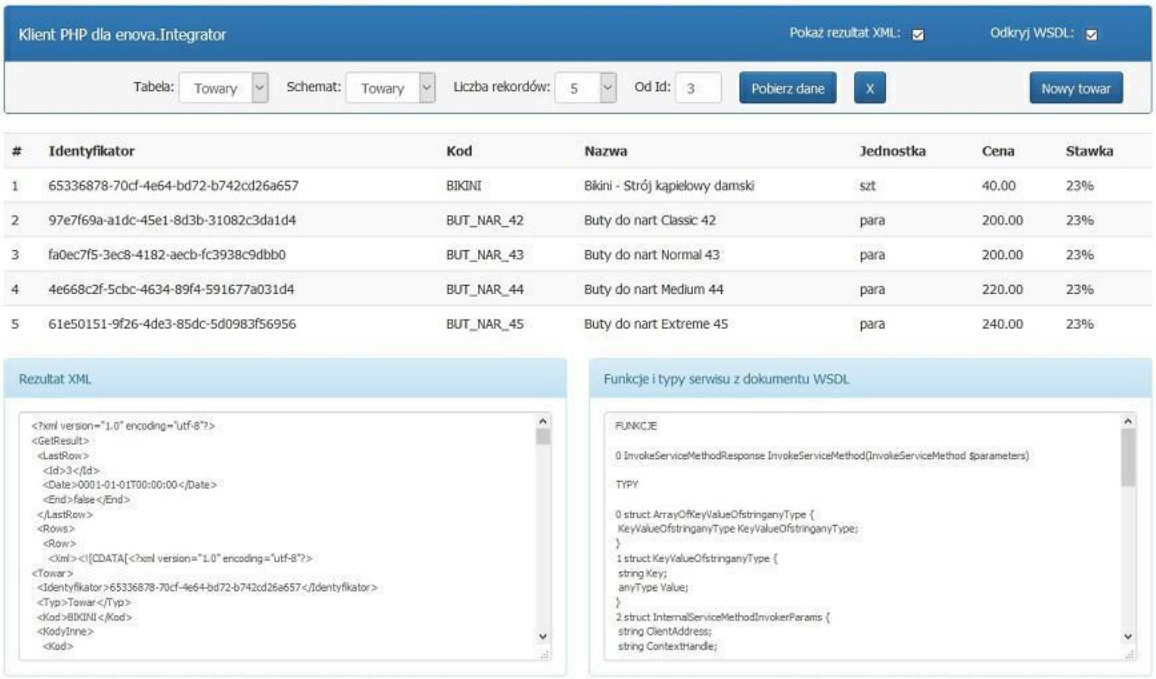
Examples of applications

Java example

For an example of a Java application using enova365.Integrator, see here. Its operation involves retrieving of one product with the Get method and then changing its name with the Update method.

PHP example

For an example of a PHP application using enova365.Integrator, see here. Its operation involves retrieving of the indicated number of goods with the Get method and displaying the data retrieved in a list in the application's window.



Example of application development

An example for development of an application using enova365.Integrator was presented during the 3rd Developers' Conference Soneta 2016. The video of the presentation: https://vimeo.com/172720412/79e504f950 .

Licence

Use of enova365.Integrator requires a licence for this module. The installation has to have a multi-licence allowing for access to the application server through the IIS website. enova365.Integrator does not accept module licences. Therefore, for logging in to enova365, it is recommended to define an operator with the assigned Task schedule role, i.e., without modules assigned.

Technical support – 12 34 92 810, techniczne@enova.pl Support HR Payroll – 12 34 92 820, place@enova.pl Support Accounting – 12 34 92 830, ksiegowosc@enova.pl Support Sales – 12 34 92 840, handel@enova.pl Support CRM – 12 34 92 850, crm@enova.pl Support Workflow & DMS – 12 34 92 860, workflow@enova.pl Support BI – 12 34 92 865, BI@enova.pl

Soneta Sp.z o.o.
ul. Wadowicka 8A, 30-415 Cracow,
Phone 12 34 92 800,
E-mail: info@enova.pl,
www.enova.pl